PROJECT 2 REPPORT

Owner: Marius THORRE, Follower: Thomas CELESCHI

Master 2 IAAA

ABSTRACT

This report delves into the concept of static word embeddings, a foundational approach in natural language processing (NLP) for representing words in a continuous vector space. Representing words in computational systems is inherently challenging, as it requires capturing both syntactic and semantic relationships in a way that machines can process efficiently. Traditional methods, like one-hot encoding, fail to account for the nuances of language, such as similarity between related words and polysemy. In this context, Word2Vec, a popular technique leveraging negative sampling, emerges as a powerful solution.

This report is structured in two main parts : the first part provides a detailed presentation of the Word2Vec method with a focus on its implementation using negative sampling. This technique enables the model to learn word representations based on their contextual similarity, making it possible to capture relationships like synonyms and analogies. In the second part, we explore various strategies to improve and analyze the basic Word2Vec model, addressing potential limitations and investigating enhancements. These include using alternative sampling methods for negative example, a embedding myth by summing different embedding together and finally a other algorithm called Fastest to add n-gram word composition into the vocabulary.

Mots clés— Word2Vec with negative sampling, word embedding, fastest

1. INTRODUCTION

In computer science, using language requires word encoding, but this is not a simple task. Therefore, a word representation must satisfy several requirements : it should enable us to analyze, compare, visualize, and sometimes understand complex objects. But how do we represent a word ? Mathematically, these representations must not be too large for computers, and linguistically, they must allow us to compute distances between words based on their meanings, such as $d(\text{tree}, \text{flower}) < d(\text{tree}, \text{building})$ and must account for polysemy, so that $\text{rep}(\text{vol}_1) \neq \text{rep}(\text{vol}_2)$. Simply using orthographic representations of words is not sufficient. A first solution is to use a *one-hot* encoding [1]. Each word is represented by a binary vector $\mathbf{v}$ whose dimension is $|V|$. Each $\mathbf{v}$ has only a 1 in the i-th position and 0 everywhere else. This method is used for simple natural language tasks, but not more because word position isn't represented distance between words in the same lexical field is lost, and finally, polysemy isn't considered. Another solution is to use the context notion. Indeed, two words that have the same context has also the same definition [2]. This report will explain this approach called Word2Vec. For the first time, a section will present this method and experimental result, code is on open source and available here : @github.com/Morpheus5828/StaticTokenDiv. Then, a second time, a presentation will be made about different techniques to improve scores and discover other ways.

2. WORD2VEC

2.1. Formulation

This method, introduced in the paper [3], addresses issues that *one-hot* encoding cannot resolve. Specifically, it links word embeddings to probability distributions through negative sampling. Negative sampling involves selecting words randomly from the vocabulary under the condition that they do not appear in the positive context of w. The algorithm's steps include generating both positive and negative samples, training a classifier to identify each type, and finally using the resulting parameters for embedding. Let w represent a randomly selected word from the vocabulary, with c as its context. The objective is to compute :

$$P(+|w, c) \tag{1}$$

In this case m and c are vectors in $\mathbb{R}^n$, it's scalary product which compute proximity between them. Then, to have number between 0 and 1, scalar result will be passed to sigmoid function :

$$\sigma(x) = \frac{1}{1 - \exp(-x)} \tag{2}$$

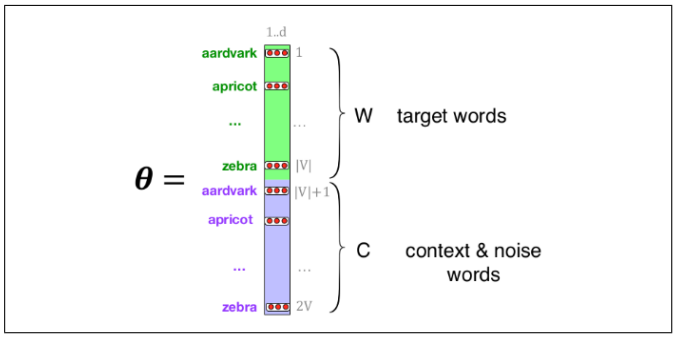
As explain before, negative sampling will be create but k times more. So equations to compute $P(+|w, c)$ and $P(-|w, c)$ will be :

$$P(+|w,c) = \sigma(m.c) = \frac{1}{1-\exp(m.c)} \tag{3}$$

$$P(-|w,c) = \sigma(m.c) = \frac{1}{1+\exp(m.c)} \tag{4}$$

To carry on on this way, a simplification is also realized, word position in context will be ignored and algo words dependencies in context.

Let's now define classifier parameters. Inputs will be two embedding matrices W and C in $\mathbb{R}^{n\times|V|}$, with n being the embedding size. They are initialized randomly and optimized for stochastic gradient descent. [4]



During each iterations, W and C will be updated about loss function define below :

$$L = -\log\left(P(+|w,c_{pos})\prod_i P(-|w,c_{neg_i})\right) \tag{5}$$

the goal is to minimize this function by Stochastic Gradient descent with η a learning rate. So during each iterations W et C will be update like this :

$$c_{pos}^{t+1} = c_{pos}^t - \eta\frac{\partial L}{\partial c_{pos}^t} \tag{6}$$

$$c_{neg}^{t+1} = c_{neg}^t - \eta\frac{\partial L}{\partial c_{neg}^t} \tag{7}$$

$$\mathbf{m}^{t+1} = \mathbf{m}^t - \eta\frac{\partial L}{\partial \mathbf{m}^t} \tag{8}$$

it's can be simplify like this :

$$c_{pos}^{t+1} = c_{pos}^t - \eta\frac{\partial L}{\partial c_{pos}^t} \tag{9}$$

$$c_{neg}^{t+1} = c_{neg}^t - \eta\frac{\partial L}{\partial c_{neg}^t} \tag{10}$$

$$\mathbf{m}^{t+1} = \mathbf{m}^t - \eta\frac{\partial L}{\partial \mathbf{m}^t} \tag{11}$$

Finally :

$$c_{pos}^{t+1} = c_{pos}^t - \eta\left[\sigma(c_{pos}\cdot\mathbf{m}) - 1\right]\mathbf{m} \tag{12}$$

$$c_{neg}^{t+1} = c_{neg}^t - \eta\left[\sigma(c_{neg}\cdot\mathbf{m})\right]\mathbf{m} \tag{13}$$

$$\mathbf{m}^{t+1} = \mathbf{m}^t - \eta\left[\sigma(c_{pos}\cdot\mathbf{m}) - 1\right]c_{pos} + \eta\sum_{i=1}^{k}\left[\sigma(c_{neg_i}\cdot\mathbf{m})\right]c_{neg_i} \tag{14}$$

Only W matrice will be keep to evaluate the computation.

2.2. Evaluation

After computing embedding a simple way to test the result is to use a script that take a test file with positive and negative context like this : **tree flower car** for example and the W matrice computed by Word2Vec and it will compute for each word m similarity with an other word in the same context called m_+ and a false context m_- :

$$sim(m,m_+) > sim(m,m_-) \tag{15}$$

A simple way to compute similiarity is to use *cosinus* function :

$$\cos(m,m_\pm) = \frac{m\cdot m_\pm}{||m||\times||m_\pm||} \tag{16}$$

2.3. Experimental Results

After implementing the Word2Vec algorithm, an initial experiment was conducted using the following parameters : $n = 100$, $k = 10$, $\eta = 0.1$, $L = 2$, $minc = 5$ (minc for minimal number occurency for each word), with 10 iterations on different corpus from author Dumas.A provided by teacher (RM for *Reine Margot*, MC for *Le compte de Monte Cristo*, V for *Le Vicomte de Bargelonne*, L3M for *Les Trois Mousquetaires* and VAA for *Vingt ans après*), evaluation script return a score of 0.60. In the next section, a presentation will be made to present different methods to upgrade this score.

3. TO GO FURTHER

3.1. Negative sampling choice

In the previous section, the negative context was selected randomly. In the paper [5], an alternative method is suggested for selecting negative samples. They assign a probability to each word in the vocabulary, calculated as the word's frequency divided by the total number of occurrences, raised to the power of α. This probability is then used to more effectively sample negative contexts based on word frequency. Score will be upgrade as show in this figure :

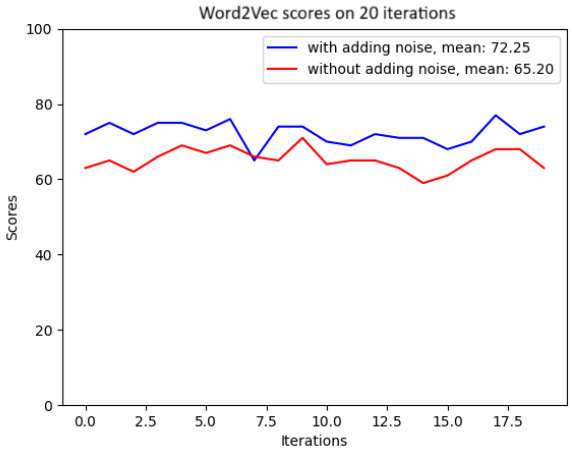
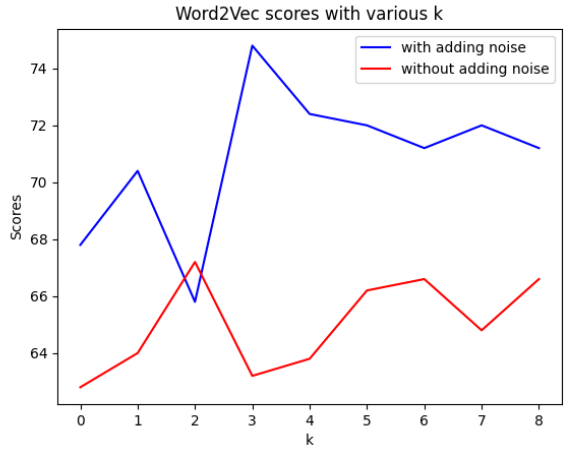


Fig. 1. Adding noise in negative sampling choice can increase score than 0.07. In this experience, k value was fix to 5 and α to 3/4.

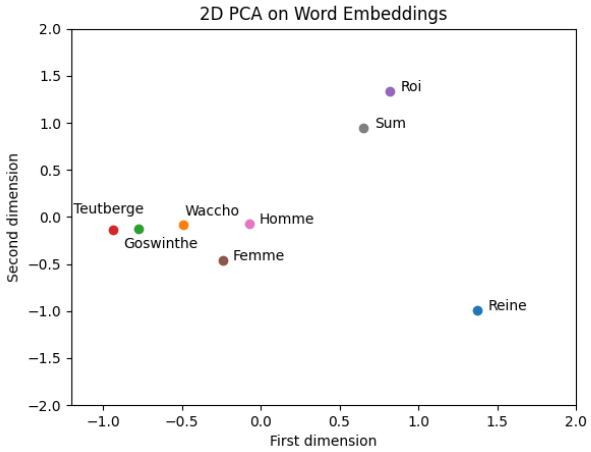
An other solution is to take different value of the k negative context and check the score. This figure below show this experience with and without adding noise :



With a k value of 3 on the addind noise method, Word2vec evaluation score is near than 0.75.

3.2. Analogy

A legend explains that it's possible to find word *reine* in French by computing *roi* - *homme* + *femme* embeddings. A demonstration of this hypothesis is available in the demos folder on the project code link (given in the Introduction). The first step consists of taking the embedding file 43 (given by the teacher) available at this link : http ://vectors.nlpl.eu/repository, and take embedding from source French words : *roi, homme* and *femme*. Compute the sum and compare the result with embedding in vocabulary. By using an algorithm to keep the k nearest neighbors of the vocabulary, apply a PCA to change the dimension from vocabulary size to 2, and finally plot it on the graph, here is the result :



With a first impression, this myth seems real, an other experience was done with taking more than one example. A file which contains 100 examples has been create for test and an algorithm implemented to compute distance between embedding sum and target word. Score is about 0.25. A conclusion can't be etablish because, different hypothesis can be think with this score. First of all, as explain before, file which contains analogies was created by programmers and not expert, and secondly corpus can be to small. By missing time, no hard answer will be given for this myth, it's work for 25 examples so it maybe not a real myth but something true. If reader want to realise experience, it's availabe by running *demo-analogy.py* file in the demo folder of code presented before.

3.3. FastText

The current method presented earlier cannot handle words not included in the vocabulary. This is particularly challenging when two words are morphologically similar, like *give* and *given* in an English corpus. A solution proposed in [6] creates embeddings for such words by decomposing them into character n-grams. For instance, *given* would be represented as : ⟨**gi, giv, ive, ven, en**⟩. An embedding is then created for each trigram, allowing the embedding for an out-of-vocabulary word to be computed as the sum of its trigrams. A demonstration of this implementation is available in the demos folder. A simple way to evaluate this task is to get the embedding of a word m and create another embedding of the same word m by computing the sum of the n-gram embeddings. For example, the word *phone* has its embedding e_m computed in the W matrix, and its trigrams : ⟨ ph, pho, hon, one, ne ⟩ are also in W. So let's define e'_m as

the embedding of m computed as :

$$e'_m = \sum_i e_{\text{gram}_i} \tag{17}$$

A comparaison can be establish to take the max between the word embedding and the word embedding created from this trigram. Calcul is the some as presented in Eq.16. In practise, Fastest need more than 5 iterations to converge. It's about 15. Actually score is near than 0.6 and by missing time, no upgrade is now available.

4. CONCLUSION

To summarize the explanations provided in this report, the introduction first outlines the topic and discusses why word embedding is an important task in NLP. The initial approach utilized one-hot encoding, however, two main problems were identified : the positions of words are not represented, and the distance between words in the same lexical field is lost. Additionally, polysemy is not considered. Subsequently, the report explores Word2Vec as a solution instead of using one-hot encoding. While Word2Vec addresses some of the limitations of one-hot encoding, it has its own constraints, such as issues related to the selection of negative sampling. In section 3.1, methods to further improve this algorithm are discussed. Finally, to advance the discussion, different phenomena related to word embedding are discovered and analyzed. One potential improvement suggested in this report is to use neural networks and deep learning libraries like PyTorch to compute the weight matrices W and C. However, due to time constraints, these methods were not implemented.

5. REFERENCES

[1] Jose Camacho-Collados and Mohammad Taher Pilehvar, "Embeddings in natural language processing," in *Proceedings of the 28th International Conference on Computational Linguistics : Tutorial Abstracts*, Lucia Specia and Daniel Beck, Eds., Barcelona, Spain (Online), Dec. 2020, pp. 10–15, International Committee for Computational Linguistics.

[2] Zellig S. Harris, "Distributional structure," *WORD*, vol. 10, no. 2-3, pp. 146–162, 1954.

[3] Tomas Mikolov, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv :1301.3781*, 2013.

[4] James H. Martin Daniel Jurafsky, "Speech and language processing," 2024.

[5] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[6] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov, "Enriching word vectors with subword information," *Transactions of the association for computational linguistics*, vol. 5, pp. 135–146, 2017.